МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РЕСПУБЛИКИ КАЗАХСТАН

Колледж Международной Академии бизнеса

Специальность 06130100 «Программное обеспечение»

Квалификация 4S06130105 «Техник информационных систем»

ОТЧЕТ

о прохождении учебно-технологической практики

Обучающегося Сумарокова Ивана Владимировича

Группы 32ПО

Место прохождения практики ТОО Smartist, г. Алматы, проспект Абая, 115

Дата начала практики «2» сентября 2024г.

Дата окончания практики «27» октября 2024г.

Руководитель от предприятия директор Сумароков Владимир Павлович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

ПЕЧАТЬ

Руководитель от колледжа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО, должность.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

Алматы, 2024г.

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| ВВЕДЕНИЕ | 3 |
| 1 ОЗНАКОМЛЕНИЕ С ПРЕДПРИЯТИЕМ |  |
| 1.1 Назначение и сетевая структура предприятия | 5 |
| 1.2 Техника безопасности и научная организация труда предприятия | 7 |
| 2 ЭТАП ПЛАНИРОВАНИЯ РАЗРАБОТКИ ТЕЛЕГРАМ-БОТА | 9 |
| 2.1 Основные цели | 9 |
| 2.2 Внешняя структура телеграм-бота | 14 |
| 2.3 Внутренняя структура телеграм-бота | 16 |
| 2.4 Инструменты создания телеграм-бота | 18 |
| 3 ЭТАПЫ РАЗРАБОТКИ ТЕЛЕГРАМ-БОТА | 20 |
| 3.1 Ключевые файлы телеграм-бота | 20 |
| 3.2 Описание работы ключевых файлов | 29 |
| 3.3 Рефакторинг файловой структуры | 35 |
| ЗАКЛЮЧЕНИЕ | 36 |
| СПИСОК ИСТОЧНИКОВ | 38 |

ВВЕДЕНИЕ

Разработка телеграм-ботов актуальна, потому что они помогают автоматизировать задачи и улучшать общение через популярный мессенджер. Боты универсальны, они могут обрабатывать заказы, управлять задачами и многое другое, экономя время и деньги компаний. Также, создание ботов дает ценный опыт, который повышает шансы найти работу в IT.

Цель практики - систематизация, закрепление и углубление теоретических знаний, полученных в процессе обучения в колледже, а также приобретение практических навыков в области проектирования и разработки программных решений для мессенджеров, в частности телеграм-ботов.

Объекты профессиональной деятельности – информационные системы и сети, информационное и программное обеспечение, разработка и эксплуатация программных решений для мессенджеров.

Практика проводилась в IT-компании ТОО Smartist, занимающейся созданием и поддержкой информационных систем, в том числе телеграм-ботов для мессенджеров. Задачи практики:

* ознакомление с работой IT-компании и основными направлениями её деятельности;
* анализ структуры и функции подразделений, занимающихся разработкой программного обеспечения;
* разработка концепции телеграм-бота, включая его функционал и возможности взаимодействия с пользователями;
* выбор инструментальных средств для разработки бота, включая язык программирования Python и фреймворки для работы с Telegram API;
* разработка и программирование функционала бота;
* интеграция бота с сторонними API;
* тестирование бота на ошибки;
* подготовка документации и материалов для отчета по практике.

1 ОЗНАКОМЛЕНИЕ С ПРЕДПРИЯТИЕМ

1.1 Назначение и сетевая структура предприятия

Компания Smartist представляет собой ведущего разработчика программного обеспечения и технологических решений. Основные цели компании включают создание инновационных систем для автоматизации бизнес-процессов, оптимизацию IT-инфраструктуры и предоставление консультационных услуг в сфере технологий.

В структуру Smartist входят несколько ключевых отделов, каждый из которых выполняет уникальные функции.

Назначение отдела разработки программного обеспечения является разработка новых программных решений, поддержка и обновление существующих продуктов.

Специфика работы этого отдела включает команды разработчиков, тестировщиков и инженеров. Основное внимание уделяется разработке качественного и безопасного ПО, а также внедрению новых технологий и методов.

Отдел проектного менеджмента координирует и управляет проектами от начала до завершения, обеспечивает соблюдение сроков и бюджета.

Специфика этого отдела включает проектных менеджеров, которые планируют проекты, определяют ресурсы, устанавливают цели и контролируют выполнение задач.

Отдел технической поддержки и обслуживания клиентов обеспечивает поддержку пользователей и решение технических проблем.

Этот отдел предоставляет помощь клиентам по вопросам эксплуатации продуктов, а также занимается решением технических проблем и вопросами технического обслуживания.

1.2 Техника безопасности и научная организация труда предприятия

В компании Smartist особое внимание уделяется технике безопасности и пожарной безопасности, чтобы обеспечить безопасные условия труда и минимизировать риски для сотрудников.

Инструктаж по технике безопасности включает обучение сотрудников основам безопасности на рабочем месте, правилам использования офисной техники и оборудования. Все новые сотрудники проходят обязательный инструктаж перед началом работы.

Специфический инструктаж проводится для работников, которые занимаются специфическими задачами, такими как работа с серверным оборудованием или системами, требующими повышенного внимания. Это включает обучение по безопасной эксплуатации специализированного ПО и оборудования.

Инструктаж по технике безопасности обновляется по мере изменения технологий или введения новых стандартов. Периодические тренинги помогают поддерживать высокий уровень осведомленности сотрудников.

Инструктаж по пожарной безопасности включает в себя практическое обучение по основам пожарной безопасности, с использованием огнетушителей и действия при обнаружении пожара и эвакуацию. Это обучение проводится при приеме на работу и регулярно повторяется.

В офисе компании размещены планы эвакуации, которые сотрудники обязаны изучить. Регулярно проводятся тренировки по эвакуации, чтобы убедиться, что все сотрудники знают свои действия в экстренных ситуациях. Ведется регулярный контроль состояния противопожарного оборудования и соблюдения норм пожарной безопасности.

Компания Smartist придает большое значение охране труда и производственной санитарии, чтобы создать здоровую и безопасную рабочую среду. Здесь обеспечивается соответствие рабочей среды современным стандартам безопасности и комфорта. В офисах устанавливаются эргономичные рабочие места, вентиляция и освещение.

Сотрудники периодически проходят медицинские осмотры для контроля состояния здоровья и предотвращения профессиональных заболеваний.

В компании предусмотрены программы поддержки психологического здоровья сотрудников, включая консультации психологов и тренинги по управлению стрессом.

В офисе поддерживается чистота на рабочих местах и в общих зонах. Регулярная уборка и дезинфекция помещений помогают предотвращать распространение заболеваний.

На предприятии организована система утилизации отходов, в том числе электроники и бумаги, с учетом экологических норм и стандартов.

Обеспечивается наличие санитарных средств, таких как антибактериальные средства для рук, и контроль за их регулярным пополнением.

ЭТАП ПЛАНИРОВАНИЯ РАЗРАБОТКИ ТЕЛЕГРАМ-БОТА

2.1 Основные цели

На этапе планирования разработки телеграм бота поставлены следующие цели:

1) Определить информационные потребности для успешного создания телеграм-бота

* + На этапе определения информационных потребностей разработчик сосредотачивается на типах данных, которые бот должен обрабатывать и передавать в канал, чтобы обеспечить участников актуальной и полезной информацией о предстоящих тренировках. Эти потребности включают название тренировки, которое должно четко идентифицировать вид занятия, позволяя участникам сразу понимать, о каком типе тренировки идет речь. Это название будет первым элементом в уведомлении, привлекающим внимание.
* Также требуется информация о дате и времени начала тренировки. Боту важно знать точное время, чтобы отправлять уведомления в нужный момент — за сутки и за час до начала мероприятия, обеспечивая тем самым своевременность и полезность уведомлений.
* Необходимы данные о месте проведения или ссылке на онлайн-платформу. Бот должен предоставлять точные координаты встречи либо виртуальную ссылку, если тренировка проходит в онлайн-формате. Это позволяет участникам легко добраться до места или подключиться к тренировке.
* Контактное лицо или пейсер тоже является важной частью информации. В уведомлении важно указать ответственного за тренировку, чтобы участники могли обратиться к нему с вопросами по мероприятию при необходимости.
* Полезно добавить описание или особые инструкции, которые включают рекомендации по экипировке, интенсивности тренировки, необходимом инвентаре или других условиях участия.
* Для уличных тренировок важными являются и погодные условия. Бот должен указывать актуальный прогноз погоды (температура, вероятность осадков, скорость и направление ветра), чтобы участники могли подготовиться к погодным условиям. Для этого бот будет интегрироваться с погодным API и запрашивать прогноз перед отправкой уведомления.

Эти информационные потребности помогают сформировать структуру уведомлений, полезных для участников и дают четкое представление о типах данных, которые должны быть доступны боту.

2) Проанализировать смежные технологии, изучить возможности интеграций

На этом этапе анализируются основные технологии, которые помогут реализовать функционал бота. Основное внимание уделяется интеграции с Telegram API для автоматической отправки уведомлений и Notion API для получения актуальной информации о запланированных тренировках. Исследование смежных технологий и альтернативных подходов к автоматизации проводится с целью обеспечения стабильной работы бота и повышения удобства его эксплуатации.

Telegram API предоставляет инструменты для создания бота, отправки сообщений в канал, а также настройки автоматизированных оповещений. Были рассмотрены возможности Telegram Bot API и его поддержка отправки сообщений с разметкой, что позволяет сделать уведомления более структурированными и удобными для восприятия. Также были проанализированы требования к токену доступа и проверена совместимость с каналом, в который бот будет отправлять уведомления.

Notion API позволяет интегрировать бот с базой данных тренировок, из которой он будет получать данные о предстоящих мероприятиях. Были изучены возможности Notion API по фильтрации данных, обеспечивающие выборку только актуальных событий. API позволяет легко получать информацию о названии тренировки, времени начала, погодных условиях и других параметрах, необходимых для формирования уведомления. В ходе анализа Notion API также проверена совместимость с используемыми типами данных, такими как дата, текст и URL.

Для получения погодных условий в уведомлениях была рассмотрена возможность интеграции с внешними сервисами, например, OpenWeather API. Этот сервис обеспечивает доступ к текущему прогнозу погоды, включая такие параметры, как температура, осадки и скорость ветра. Его использование позволит боту запрашивать актуальный прогноз для уведомлений, отправляемых за сутки и за час до тренировки.

В ходе анализа были рассмотрены альтернативные подходы к автоматизации, такие как использование Google Calendar API для управления календарем и отправки напоминаний. Однако с учетом требований проекта и структуры данных в Notion, было решено, что Notion API более точно отвечает целям бота и позволяет избежать дублирования данных в различных сервисах.

Проведенный анализ позволил утвердить выбор инструментов и API, которые обеспечат стабильную и эффективную работу бота для уведомления о тренировках в Telegram-канале.

3) Постановка задачи

Задача разработки телеграм-бота заключается в создании автоматизированной системы оповещений о предстоящих тренировках. Бот должен извлекать данные из базы тренировок в Notion и отправлять соответствующие уведомления в указанный Telegram-канал за определенное время до начала мероприятия.

Основные задачи включают:

* Автоматизация отправки уведомлений. Бот должен автоматически отправлять сообщения в телеграм-канал за сутки и за час до начала тренировки. Эти сообщения должны содержать информацию о названии тренировки, дате, времени, месте, погодных условиях и прочих инструкциях для участников.
* Интеграция с Notion API. Бот должен подключаться к базе данных Notion, где хранятся записи о тренировках. Ему нужно считывать необходимые данные (название, дата, время, место, контактное лицо и погодные условия) и формировать на их основе уведомления.
* Подключение к Telegram API. Система должна иметь возможность отправлять уведомления в указанный Telegram-канал, используя токен доступа к Telegram API, и предоставлять сообщения с четкой структурой и оформлением.
* Интеграция с погодным API. Для уличных тренировок бот должен добавлять актуальные погодные условия, такие как температура, осадки и ветер, используя погодный API (например, OpenWeather). Запрос прогноза погоды должен выполняться непосредственно перед отправкой уведомления.
* Надежность и отказоустойчивость. Бот должен корректно обрабатывать запросы к API и обеспечивать устойчивость к сетевым задержкам, чтобы информация о тренировках отправлялась вовремя. При необходимости бот должен повторять запросы, если данные Notion или погодного API временно недоступны.
* Удобная настройка и расширяемость. Задача предполагает создание кода, который будет легко масштабировать и изменять. Например, возможно добавление новых типов уведомлений или расширение функционала бота.

В итоге, телеграм-бот должен представлять собой устойчивую, интегрированную с внешними сервисами систему, которая автоматически отправляет уведомления, обеспечивая участников актуальной информацией и упрощая процесс подготовки к тренировкам.

2.2 Внешняя структура бота

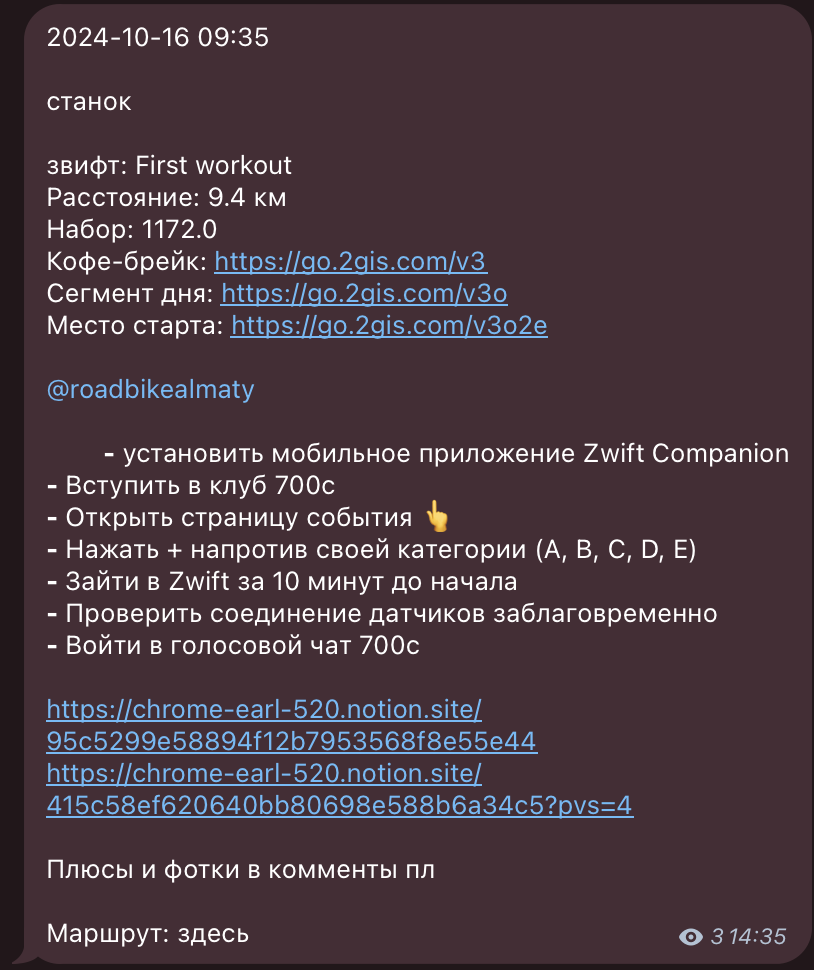
Архитектура внешней структуры телеграм-бота выглядит следующим образом:

1) Определение основных блоков телеграм-бота

* Планировщик задач операционной системы сервера
* Docker контейнер для запуска приложения
* Модуль \_\_main\_\_ - инициализация классов и запуск метода основного класса SendAnounce
* Классы получения данных API
* Классы создания моделей (Ride, Route и т.д.)
* Класс подготовки текста сообщения AnounceView

2) Комбинирование блоков в следующем порядке

* команда операционной системы Cron (планировщик задач)
* Запуск Docker-контейнера
* Выполнение приложения внутри контейнера
* Запросы к API (Notion, OpenWeather) и получение данных
* Обработка данных
* Формирование сообщения
* Отправка сообщения в Telegram-канал

Предварительный графический макет

2.3 Внутренняя структура бота

Внутренняя структура телеграм-бота организована таким образом, чтобы обеспечивать поддерживаемость, гибкость и расширяемость кода. Основой структуры является построение четких связей между модулями и использование принципов чистого кода, предложенных Робертом Мартином. Этот подход помогает создать код, который легко читать, понимать и поддерживать. Архитектура приложения ориентирована на минимизацию сложностей и разделение ответственности, что делает систему более надежной и гибкой к изменениям.

Построение структуры ссылочных связей

Для достижения этих целей применяется методология чистого кода и принципы SOLID. Принципы SOLID являются руководящими правилами для проектирования качественного и поддерживаемого кода.

* Single Responsibility Principle (SRP) — принцип единственной ответственности. Каждый класс или модуль должен иметь одну четко определенную ответственность. В контексте бота это означает, что каждый компонент, например, модуль для работы с Telegram API или модуль для получения данных из Notion, отвечает только за свою функциональность. Это упрощает отладку и тестирование.
* Open/Closed Principle (OCP) — принцип открытости/закрытости. Модули или классы должны быть открыты для расширения, но закрыты для модификации. Если возникает необходимость добавить новую функциональность, код можно расширить без внесения изменений в существующий код. В боте, например, можно добавить новые типы уведомлений, не изменяя уже существующие модули.
* Liskov Substitution Principle (LSP) — принцип подстановки Барбары Лисков. Объекты подклассов должны быть взаимозаменяемы с объектами базового класса без нарушения работы программы. Это упрощает замену модулей и компонентов, если возникает необходимость в изменении структуры бота.
* Interface Segregation Principle (ISP) — принцип разделения интерфейса. Лучше создавать несколько специализированных интерфейсов для каждого типа задачи, чем один универсальный. Это позволяет боту иметь интерфейсы, ориентированные только на необходимые функции, обеспечивая компактность и упрощая поддержку.
* Dependency Inversion Principle (DIP) — принцип инверсии зависимостей. Модули высокого уровня не должны зависеть от модулей низкого уровня; оба должны зависеть от абстракций. В случае с ботом, это помогает минимизировать жесткие зависимости между компонентами, делая систему более гибкой.

Применение принципов SOLID и методологии чистого кода помогает структурировать бота так, чтобы каждый модуль был легко читаемым, тестируемым и независимым от других модулей.

Построение файловой структуры (Приложение)

2.4 Инструменты создания бота

Для разработки телеграм-бота был выбран язык Python, который, благодаря большому набору библиотек и простоте синтаксиса, позволяет эффективно реализовывать функционал без значительной нагрузки на ресурсы системы. Python подходит для быстрого прототипирования и легко поддерживается за счет доступности множества специализированных инструментов и библиотек.

В процессе разработки использовался NeoVim с набором полезных плагинов, выполняющих различные функции, что облегчает работу над кодом:

Yazi — для управления файлами и организации файловой структуры проекта.

Pytest — для запуска и отладки тестов, что помогает поддерживать стабильность кода.

Ruff — линтер, обеспечивающий автоматическое форматирование кода и соблюдение стандартов стиля.

Mypy — статический анализатор, позволяющий выявлять ошибки типов еще на этапе разработки.

Для реализации ключевого функционала использовались следующие библиотеки:

Requests — для создания и выполнения HTTP-запросов к API Notion, что позволяет получать данные о тренировках.

PyTelegramBotAPI — для упрощенной работы с Telegram API, включая отправку уведомлений в канал.

Pydantic — для валидации и работы с JSON-данными, что помогает проверять корректность данных, поступающих из API.

Thunder Client — расширение для Visual Studio Code, которое упрощает тестирование HTTP-запросов, предоставляя удобный интерфейс для работы с API.

Этот набор инструментов и библиотек позволяет поддерживать структурированность, качество и легкость поддержки кода, а также ускоряет процесс разработки и тестирования телеграм-бота.

3. ЭТАПЫ РАЗРАБОТКИ ТЕЛЕГРАМ-БОТА

3.1 Ключевые модули телеграм-бота

Для разработки телеграм бота мы использовали два основных модуля ride и route.

1) Модуль ride предназначен для парсинга данных из сервиса Notion и заполнения этими данными класса ride, который играет важную роль в телеграм-боте, предоставляя пользователям актуальную информацию о предстоящих заездах и тренировках.

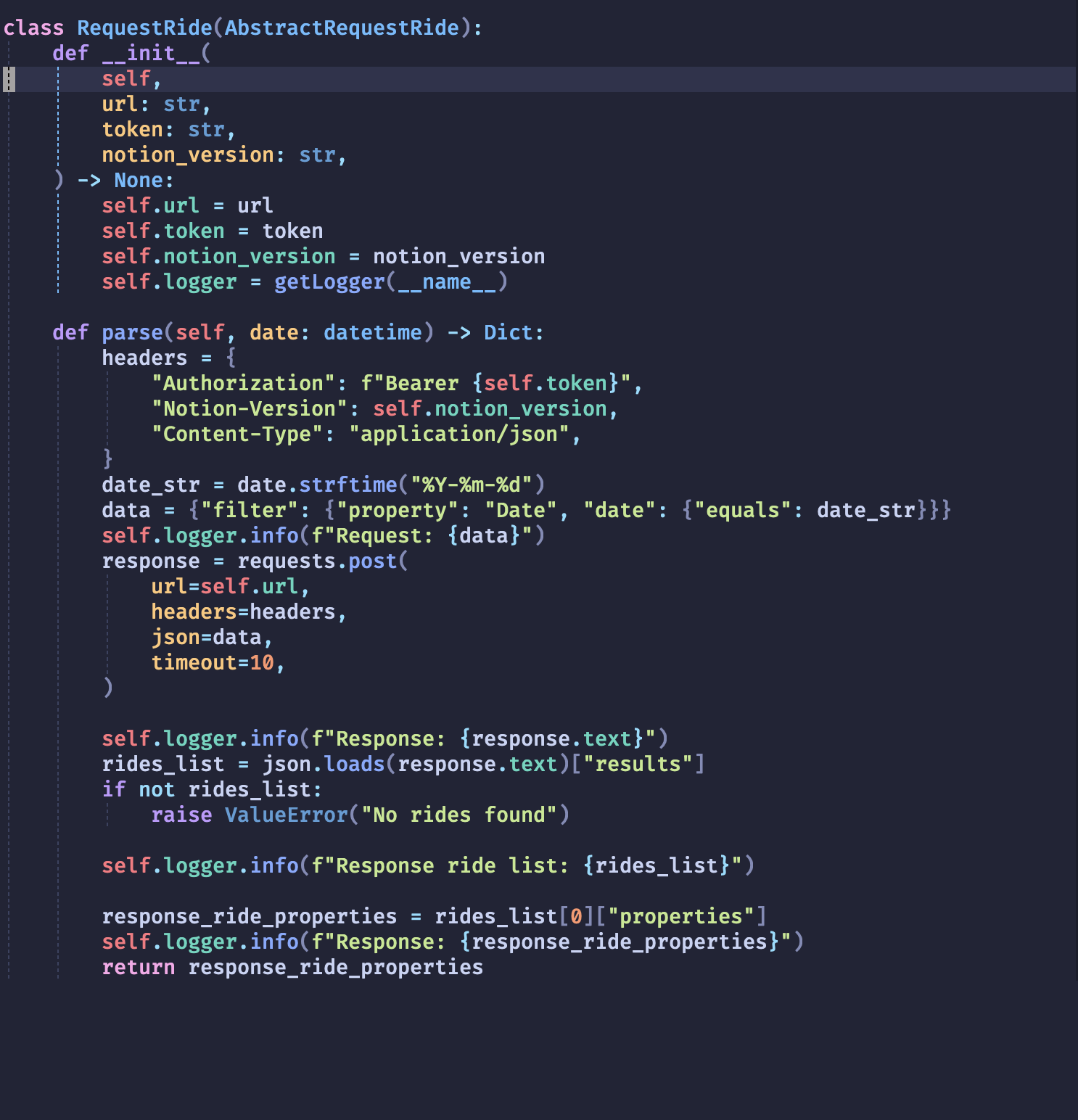
Класс CreateRide наследуется от AbstractCreateRide и отвечает за создание экземпляра класса Ride, используя данные, полученные из Notion.

Данные принимает словарь ride\_properties, содержащий свойства заезда, извлеченные из Notion. Извлечение и обработка данных происходит из поля ride\_properties

С Notion получаем ключи

* + - \_type\_of\_training: тип тренировки (из ride\_properties["Тип тренировки"]["select"]["name"]).
    - \_date: дата заезда (из ride\_properties["Date"]["date"]["start"]).
    - \_route\_id: идентификатор маршрута (из ride\_properties["Маршрут"]["relation"][0]["id"]).
    - \_voice\_channel: ссылка на голосовой канал (из ride\_properties["voice\_channel"]["url"]).
    - \_pacer: имя пейсера или лидера заезда (из ride\_properties["Pacer"]["people"][0]["name"]).
    - \_power: мощность или интенсивность заезда (из ride\_properties["мощьность"]["select"]["name"]).
    - \_post\_scriptum: дополнительная информация (если имеется, иначе пустая строка).
    - \_workout\_app: используемое приложение для тренировки (из ride\_properties["application"]["select"]["name"]).
    - \_condition: условия участия (если имеются, иначе пустая строка).
    - \_gestures: ссылка на жесты или сигналы (из ride\_properties["gest ures"]["url"]).
    - \_manual\_for\_payser: руководство для пейсера (из ride\_properties["manual\_for\_payser"]["url"]).
    - \_name\_workout: название тренировки или заезда (из ride\_properties[“Name”][“title"][0]["text"]["content"]).

Создание объекта Ride дает нам экземпляр класса Ride, передавая извлеченные данные в качестве параметров конструктора. Возврат результата возвращает созданный объект Ride для дальнейшего использования в приложении.

Класс RequestRide наследуется от AbstractRequestRide и отвечает за получение данных о заездах из базы данных Notion по заданной дате.

При запуске (инициализации) принимает параметры url (базовый URL для обращения к API Notion), token (токен доступа для авторизации в Notion) и notion\_version (версия API Notion). Инициализирует логгер для отслеживания процесса выполнения.

Метод parse(date: datetime) Dict: используем для формирование запроса. Он устанавливает заголовки запроса с необходимыми параметрами авторизации и версии API. Преобразует дату в строковый формат YYYY-MM-DD. Создает тело запроса с фильтром по дате для получения заездов, запланированных на указанную дату.

Взаимодействие классов

RequestRide получает актуальные данные о заездах из Notion по заданной дате.

CreateRide принимает данные от RequestRide и создает структурированный объект Ride.

Модуль ride обеспечивает важную функционциональность нашего телеграм-бота, позволяя автоматически получать и обрабатывать информацию о предстоящих заездах из Notion. Благодаря классам CreateRide и RequestRide, мы можем преобразовывать данные из внешнего сервиса в удобный формат для пользователей. Структура модуля с разделением на абстрактные и конкретные реализации упрощает поддержку и дальнейшее развитие приложения. Использование логирования и обработки исключений повышает надежность и упрощает отладку кода.

2) Модуль route включает в себя два ключевых класса CreateRoute и ParseRouteData, которые совместно обеспечивают парсинг данных из Notion и создание объектов маршрутов для использования в нашем телеграм-боте.

Класс CreateRoute

Наследуется от AbstractCreateRoute и отвечает за создание экземпляра класса Route, заполняя его данными, полученными из Notion.

Метод create\_route(route\_properties: dict) класса CreateRoute

Данные принимает словарь route\_properties, содержащий свойства маршрута, извлеченные из Notion.

Извлечение и обработка данных

 извлекает следующие поля из route\_properties

* + - \_day\_segment: сегмент дня (из route\_properties.get("сегмент дня ", {}).get("url")).
    - \_start\_place: место старта (из route\_properties.get("Место старта ", {}).get("url")).
    - \_set\_count: набор высоты в метрах (из route\_properties.get("Набор, м", {}).get("number")).
    - \_coffee\_break: информация о кофе-брейке (из route\_properties.get("кофе брейк ", {}).get("url")).
    - \_distance: дистанция маршрута в километрах (из route\_properties.get("Дистанция", {}).get("number")).
    - \_route\_name: название маршрута (извлекается через цепочку методов для получения текста из заголовка).

Создание объекта Route

Метод \_\_init\_\_ создает экземпляр класса Route, передавая извлеченные данные в качестве параметров конструктора.Возврат результата возвращает созданный объект Route для дальнейшего использования в приложении.

Класс ParseRouteData

Наследуется от AbstractParseRoute и отвечает за получение данных о маршрутах из базы данных Notion.

Инициализация - принимает параметры notion\_base\_url и notion\_token, необходимые для авторизации и взаимодействия с API Notion. Сохраняет эти параметры как атрибуты класса для последующего использования.

Метод parse(route\_database\_id: str) -> dict:

Формирование запроса

* + - Формирует URL запроса к API Notion с использованием route\_database\_id.
    - Устанавливает заголовки запроса, включая авторизацию, версию API Notion и тип контента.

Выполнение запроса

* + - Отправляет GET-запрос к API Notion для получения данных страницы маршрута.
    - Логирует заголовки и ответ для отладки.

Обработка ответа

* + - Проверяет статус-код ответа; если он не равен 200, выбрасывает исключение с соответствующим сообщением об ошибке.
    - Парсит тело ответа из формата JSON и извлекает свойства маршрута.
    - Если свойства не найдены, выбрасывает исключение с сообщением "Route not found".

Возврат результата: возвращает словарь route\_properties, содержащий данные о маршруте, для последующей обработки в классе CreateRoute.

Взаимодействие классов

ParseRouteData получает сырые данные о маршруте из Notion, обеспечивая актуальность и точность информации.

CreateRoute принимает эти данные и создает объект Route, который готов к использованию в боте.

Таким образом, совместная работа этих классов позволяет эффективно парсить данные из Notion и преобразовывать их в удобный для использования формат. Это обеспечивает пользователей актуальной информацией о маршрутах и улучшает общий опыт взаимодействия с приложением.

Примечание

Код содержит механизмы логирования для отслеживания процесса выполнения и отладки. Используются проверки на наличие данных и корректность ответа от API, что повышает надежность работы приложения. Структура кода и его разделение на абстрактные и конкретные классы упрощает поддержку и возможность расширения функциональности в будущем.

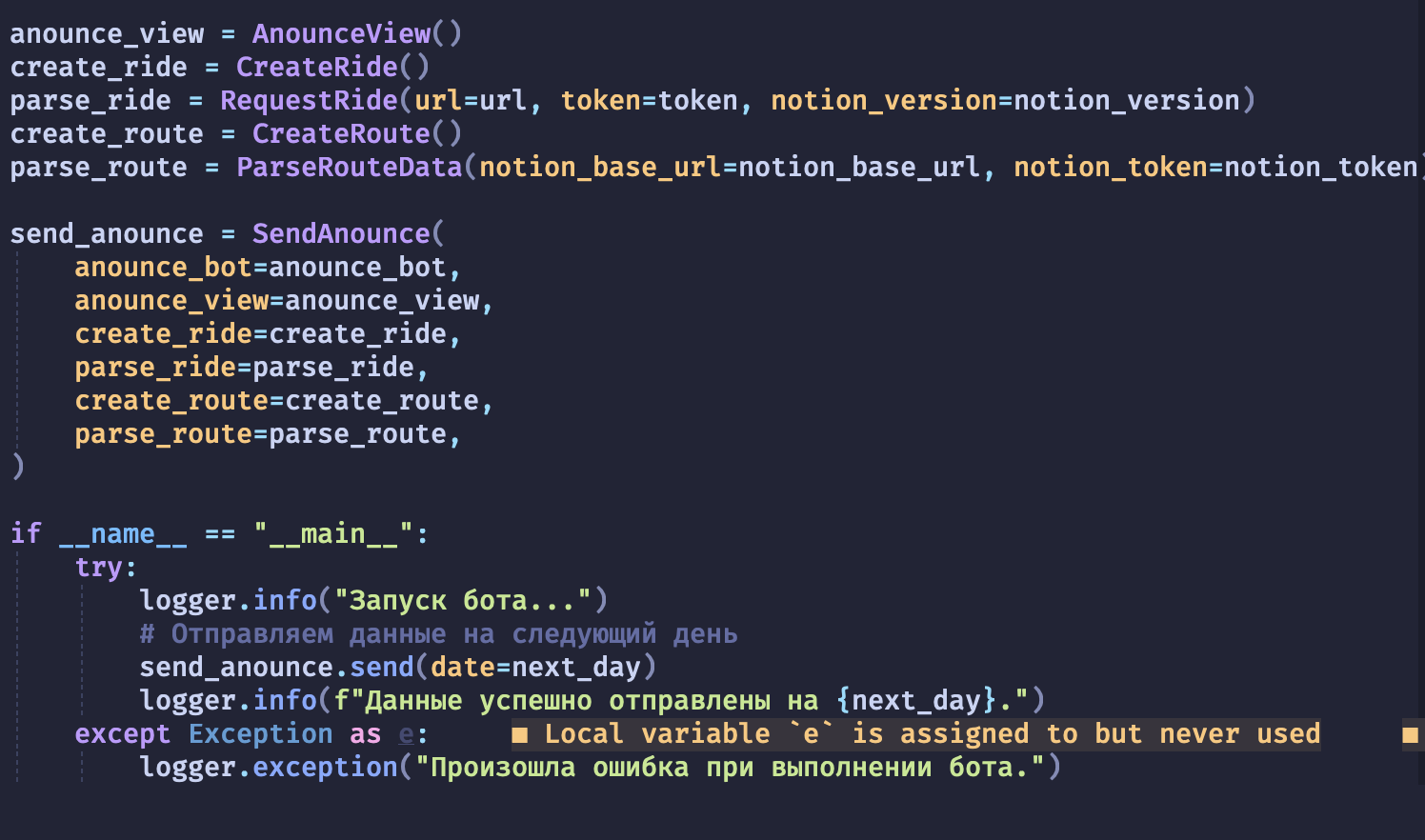
3.2 Описание работы ключевых файлов

Ключевые файлы телеграм-бота

send\_anounce.py

Класс SendAnounce предназначен для отправки анонсов через телеграм-бота. Он наследуется от абстрактного класса AbstractSend и включает несколько компонентов, которые предоставляют различные функции для создания и обработки маршрутов и поездок.

Содержит следующие компоненты

* anounce\_bot: экземпляр класса TeleBot, представляющий телеграм-бота для отправки сообщений.
* anounce\_view: используется для представления информации о поездке в формате текста.
* create\_ride: предназначен для создания объекта поездки на основе свойств поездки.
* parse\_ride: отвечает за извлечение (парсинг) свойств поездки.
* create\_route: используется для создания маршрута.
* parse\_route: предназначен для извлечения свойств маршрута.
* Метод send(self, date: datetime) -> None отправляет анонс поездки на указанный канал телеграм:
* Принимает дату и использует parse\_ride для получения свойств поездки.
* Создает объект поездки с помощью create\_ride.
* Использует parse\_route для получения свойств маршрута по ID маршрута.
* Создает маршрут с помощью create\_route и добавляет его в объект поездки.
* Генерирует текст сообщения с помощью anounce\_view.
* Отправляет сообщение в канал, ID которого получен из переменной окружения CHANNEL\_ID, используя anounce\_bot.send\_message() с HTML-разметкой.
* Этот класс модульно организует отправку анонсов, позволяя легко изменять логику создания и обработки данных для поездок и маршрутов.
* 2 main.py

Основной скрипт для запуска телеграм-бота, который взаимодействует с API Notion для отправки анонсов.

Основные блоки и их настройка

Загрузка переменных окружения. Переменные из .env загружаются с помощью load\_dotenv. Эти переменные содержат токены и идентификаторы, необходимые для работы с Telegram и Notion API.

Логи настраиваются с помощью basicConfig и FileHandler. Логи записываются в файл bot.log, а конфигурация включает уровни логирования и формат записей.

Инициализация компонентов

* + anounce\_bot: экземпляр класса TeleBot, инициализируется с токеном из переменной окружения BOT\_TOKEN.
  + calendar\_url и calendar\_id: составляют URL календаря Notion для получения данных о событиях.
  + notion\_token и notion\_base\_url: используются для подключения к Notion API.

Создание экземпляров вспомогательных классов: классы AnounceView, CreateRide, RequestRide, CreateRoute, ParseRouteData и SendAnounce отвечают за логику работы с событиями и маршрутами.

Основная логика запуска:

В блоке if \_\_name\_\_ == "\_\_main\_\_": происходит:

* + Логирование о запуске бота.
  + Вызов метода send() для отправки анонсов на следующий день.
  + Логирование успешной отправки данных или запись ошибки при возникновении исключений.

3 anounce\_view.py

Модуль AnounceView отвечает за формирование текстового сообщения для анонса поездки в нашем телеграм-боте.

Класс AnounceView

Пользовательский класс исключения, наследующийся от AbstractView. Используется для обработки ситуации, когда информация о маршруте отсутствует в объекте Ride.

Класс AnounceView

Создает информативное текстовое сообщение для пользователей, используя данные из объекта Ride.

Метод get(self, ride: Ride) -> str:

Получает текущее время в формате UTC и форматирует его в строку вида YYYY-MM-DD HH:MM.

Проверяет наличие маршрута в объекте ride; если его нет, выбрасывает исключение AnounceView.

Формирует текстовое сообщение, включающее:

* + - Название тренировки.
    - Текущее время.
    - Тип тренировки.
    - Приложение для тренировки.
    - Данные маршрута: расстояние, набор высоты, кофе-брейк, сегмент дня, место старта.
    - Ссылку на голосовой канал.
    - Условия участия, руководство для пейсера, жесты и сигналы.
    - Дополнительную информацию.
    - Ссылку на маршрут.
    - Использует вспомогательный метод \_prepare\_text для корректного отображения данных.

Метод \_prepare\_text(self, data: str | float | None) -> str:

Обрабатывает входящие данные перед вставкой в сообщение, чтобы избежать ошибок и некорректного отображения.Если данные отсутствуют, возвращает "-".Если данные типа float, округляет до двух знаков после запятой.Если данные типа str, заменяет "None" на "-".

Разработанный файл обеспечивает генерацию структурированного и информативного анонса поездки для пользователей нашего бота. Он учитывает все необходимые детали поездки и маршрута, форматирует их должным образом и обеспечивает обработку возможных исключений. Это значительно улучшает пользовательский опыт, делая информацию доступной и понятной.

3.3 Рефакторинг файловой структуры для упрощения поддержки кода

* Создание отдельных папок для абстрактных классов.
* Создание папок для доменных объектов и анализ кода при помощи инструментов ruff и mypy.
* Покрытие кода тестами.
* Тестирование и отладка.
* Развертывание (деплой) приложения.

ЗАКЛЮЧЕНИЕ

В ходе прохождения практики в IT-компании ТОО Smartist были успешно достигнуты поставленные цели и решены все задачи, обозначенные во введении. Практика позволила систематизировать, закрепить и углубить теоретические знания, полученные в колледже, а также приобрести практические навыки в области проектирования и разработки программных решений для мессенджеров, в частности телеграм-ботов.

Выполнение поставленных задач:

* Успешно прошло ознакомление со структурой компании Smartist, ее основными направлениями работы и принципами функционирования. Это дает общее представление о том, как организована работа в современной IT-компании.
* Анализ структуры и функций подразделений, занимающихся разработкой программного обеспечения позволило изучить организацию работы отделов разработки, роли и обязанности сотрудников, участвующих в процессе создания программного обеспечения. Это помогло понять важность командной работы и эффективного распределения задач.
* Разработка концепции телеграм-бота, включая его функционал и возможности взаимодействия с пользователями. Была разработана концепция бота, определены его основные функции и сценарии взаимодействия с пользователями. Особое внимание уделялось удобству использования и полезности бота для конечных пользователей.
* Выбор инструментальных средств для разработки бота, включая язык программирования Python и фреймворки для работы с Telegram API. Проведён анализ различных инструментов и библиотек. В результате для разработки был выбран язык Python и фреймворк PyTelegramBotAPI, что обеспечило быструю и эффективную реализацию функционала бота.
* Разработка и программирование функционала бота. Реализован функционал бота согласно разработанной концепции. Бот способен обрабатывать запросы пользователей, взаимодействовать с API и предоставлять необходимую информацию.
* Интеграция бота с базами данных и сторонними API. Бот успешно интегрирован с базой данных и внешним API сервиса Notion. Это позволило автоматизировать процесс получения и обработки данных, необходимых для работы бота.
* Тестирование бота на ошибки. Проведено тщательное тестирование бота в различных сценариях использования. Обнаруженные ошибки были исправлены, что повысило стабильность и надёжность работы приложения.
* Подготовка документации и материалов для отчёта по практике. Разработана полная документация по проекту, включая описание архитектуры бота, его функционала и инструкции по развертыванию. Это облегчает дальнейшую поддержку и развитие проекта.

Созданный телеграм-бот автоматизирует процесс рассылки уведомлений о событиях, что экономит время и ресурсы компании. Он улучшает взаимодействие с пользователями и повышает эффективность коммуникации. Полученные во время практики знания и навыки в области разработки ботов и интеграции с внешними сервисами являются ценными для профессионального развития и повышают шансы на успешную карьеру в сфере IT.

Практика была крайне полезной и продуктивной. Поставленные цели достигнуты, получен реальный опыт работы в команде, освоены современные технологии и инструменты разработки телеграм-бота. Это можно считать основой для дальнейшего профессионального роста и развития в сфере информационных технологий.

СПИСОК ИСТОЧНИКОВ

1. Документация Ruff. Инструмент для анализа кода Python [Электронный ресурс] — Режим доступа: <https://beta.ruff.rs/docs/> (дата обращения: 15.09.2024).
2. Лутц М. Изучаем Python, 5-е издание. — М.: "Вильямс", 2020. — 1500 с.
3. Макконнелл С. Совершенный код. Мастер-класс. — М.: "Русская Редакция", 2019. — 896 с.
4. Мартин Р. К. Чистый код: создание, анализ и рефакторинг. — М.: "Питер", 2019. — 464 с.
5. Официальная документация Notion API [Электронный ресурс] — Режим доступа: <https://developers.notion.com/> (дата обращения: 20.09.2024).
6. Официальная документация Telegram Bot API [Электронный ресурс] — Режим доступа: <https://core.telegram.org/bots/api> (дата обращения: 24.09.2024).
7. Mypy  Статический типизатор для Python [Электронный ресурс] — Режим доступа: <http://mypy-lang.org/> (дата обращения: 29.09.2024).
8. PEP 8  Стандарт оформления кода на Python [Электронный ресурс] — Режим доступа: <https://peps.python.org/pep-0008/> (дата обращения: 02.10.2024).
9. Pydantic  Библиотека для валидации данных и управления настройками [Электронный ресурс] — Режим доступа: <https://pydantic-docs.helpmanual.io/> (дата обращения: 02.10.2024).
10. pyTelegramBotAPI  Библиотека для работы с Telegram Bot API на Python [Электронный ресурс] — Режим доступа: <https://github.com/eternnoir/pyTelegramBotAPI> (дата обращения: 19.09.2024).
11. Requests: HTTP for Humans Библиотека для работы с HTTP-запросами на Python [Электронный ресурс] — Режим доступа: <https://requests.readthedocs.io/> (дата обращения: 4.10.2024).
12. Pytest Фреймворк для тестирования на Python [Электронный ресурс]— Режим доступа: <https://docs.pytest.org/> (дата обращения: 11.10.2024).
13. SOLID Principles  Принципы объектно-ориентированного программирования [Электронный ресурс]— Режим доступа: <https://ru.wikipedia.org/wiki/SOLID> (дата обращения: 13.10.2024).
14. Thunder Client  Расширение для Visual Studio Code для тестирования API [Электронный ресурс] — Режим доступа: <https://www.thunderclient.com/> (дата обращения: 16.10.2024).

Приложение

Структура файлов проекта

├── Dockerﬁle

├── compose.yaml

├── json\_ﬁles

├── poetry.lock

├── project\_ﬁles\_tree.md

├── pyproject.toml

├── requirements.txt

├── src

│   ├── \_\_init\_\_.py

│   ├── anounce

│   │   ├── \_\_init\_\_.py

│   │   ├── \_\_main\_\_.py

│   │   ├── abs

│   │   │   ├── \_\_init\_\_.py

│   │   │   ├── abc\_send.py

│   │   │   └── abc\_view.py

│   │   ├── anounce\_view.py

│   │   ├── ride

│   │   │   ├── \_\_init\_\_.py

│   │   │   ├── abs

│   │   │   │   ├── \_\_init\_\_.py

│   │   │   │   ├── abc\_create\_ride.py

│   │   │   │   └── abc\_request\_ride.py

│   │   │   ├── implementations

│   │   │   │   ├── \_\_init\_\_.py

│   │   │   │   ├── create\_ride.py

│   │   │   │   └── request\_ride.py

│   │   │   └── model.py

│   │   ├── route

│   │   │   ├── \_\_init\_\_.py

│   │   │   ├── abs

│   │   │   │   ├── \_\_init\_\_.py

│   │   │   │   ├── abc\_create\_route.py

│   │   │   │   ├── abc\_parse\_route.py

│   │   │   │   └── abc\_route\_provider.py

│   │   │   ├── implementations

│   │   │   │   ├── \_\_init\_\_.py

│   │   │   │   ├── create\_route.py

│   │   │   │   └── parse\_route.py

│   │   │   └── model.py

│   │   ├── send\_anounce.py

│   │   ├── test.json

│   │   └── tests

│   │       ├── \_\_init\_\_.py

│   │       ├── test\_anounce\_viev.py

│   │       ├── test\_create\_ride.py

│   │       ├── test\_create\_route.py

│   │       └── test\_parse\_route.py

│   ├── bot

│   │   ├── \_\_main\_\_.py

│   ├── handlers

│   │   ├── \_\_init\_\_.py

│   │   ├── callbackdata\_participate\_button\_1\_1.py

│   │   └── start\_message\_handler.py

│   ├── keyboards

│   │   ├── \_\_init\_\_.py

│   │   └── seen.py

│   ├── models

│   │   ├── bot

│   │   │   ├── \_\_init\_\_.py

│   │   │   ├── keyboard.py

│   │   │   ├── message\_handler.py

│   │   │   └── view.py

│   │   └── notion\_calendar

│   └── parse\_calendar

**Характеристика - отзыв**

**о прохождении учебно-технологической практики**

**студента(ки)**

**Колледжа Международной Академии бизнеса**

Обучающийся Сумароков Иван Владимирович 32 ПО

проходил практику с 2 сентября 2024 г. по 27 октября 2024 г.

на ТОО Smartist в подразделении отдел бэкхенд разработки

За период прохождения практики практикант посетил 38 дней, из них по уважительной причине отсутствовал 2 дней, пропуски без уважительной причины составили - дней.

Практикант соблюдал/не соблюдал трудовую дисциплину и /или правила техники безопасности.

Отмечены нарушения трудовой дисциплины и /или правил техники безопасности: нет

Практикант не справился со следующими видами работ: —

за время прохождения практики показал, что *(подчеркнуть нужное)*

Сумароков Иван

что умеет/не умеет планировать и организовывать собственную деятельность, способен/не способен налаживать взаимоотношения с другими сотрудниками, имеет/не имеет хороший уровень культуры поведения, умеет/не умеет работать в команде, высокая/низкая степень сформированности умений в профессиональной деятельности.

В отношении выполнения трудовых заданий проявил себя хорошими навыками взаимодействия в команде; вносил предложения и был полезен. Иван быстро осваивал новые технологии и быстро адаптировался к изменениям. С поставленной задачей справился полностью, разработав документ с нуля до деплоя (вспомогательный телеграм-бот).

Директор Сумароков В.П.

*Должность руководителя практики подпись                                                         И.О. Фамилия*